NOM

ip − Implémentation Linux du protocole IPv4.

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h> /* super ensemble du précédent */

tcp_socket = **socket(PF_INET, SOCK_STREAM, 0);**
udp_socket = **socket(PF_INET, SOCK_DGRAM, 0);**
raw_socket = **socket(PF_INET, SOCK_RAW,** *protocol***);**

DESCRIPTION

Linux implémente le Protocole Internet (IP) version 4, décrit dans les RFC 791 et RFC 1122. **ip** contient une implémentation du multicasting niveau 2 conforme à la RFC 1112. Elle contient aussi un routeur IP comprenant un filtre de paquets.

L'interface de programmation est compatible avec les sockets BSD. Pour plus de renseignements sur les sockets, voir **socket**(7).

Une socket IP est créée par la fonction **socket**(2) invoquée sous la forme **socket(PF_INET, type_socket, protocole)**. les types valides des sockets sont **SOCK_STREAM** pour ouvrir une socket **tcp**(7), **SOCK_DGRAM** pour ouvrir une socket **udp**(7), ou **SOCK_RAW** pour ouvrir une socket **raw**(7) permettant d'accéder directement au protocole IP. Le *protocole* indiqué est celui inscrit dans les en-têtes IP émis ou reçus. Les seules valeurs valides pour le *protocole* sont **0** et **IPPROTO_TCP** pour les sockets TCP, et **0** et **IPPROTO_UDP** pour les sockets UDP. Pour les sockets **SOCK_RAW** on peut indiquer un protocole IP IANA valide dont la RFC 1700 précise les numéros assignés.

Lorsqu'un processus veut recevoir de nouveaux paquets entrants ou connexions, il doit attacher une socket à une adresse d'interface locale en utilisant **bind**(2). Une seule socket IP peut être attachée à une paire (adresse, port) locale donnée. Lorsqu'on indique **INADDR_ANY** lors de l'attachement, la socket sera affectée à *toutes* les interfaces locales. Si on invoque **listen**(2) ou **connect**(2) sur une socket non affectée, elle est automatiquement attachée à un port libre aléatoire, avec l'adresse locale fixée sur **INADDR_ANY**.

L'adresse locale d'une socket TCP qui a été attachée est indisponible pendant quelques instants après sa fermeture, à moins que l'attribut **SO_REUSEADDR** ait été activé. Il faut être prudent en utilisant ce drapeau, car il rend le protocole TCP moins fiable.

FORMAT D'ADRESSE

Une adresse de socket IP est définie comme la combinaison d'une adresse IP d'interface et d'un numéro de port 16 bits. Le protocole IP de base ne fournit pas de numéro de port, ils sont implémentés par les protocoles de plus haut-niveau comme **udp**(7) et **tcp**(7). Sur les sockets raw, le champ *sin_port* contient le protocole IP.

```
struct sockaddr_in {
    sa_family_t    sin_family; /* famille d'adresses : AF_INET     */
    u_int16_t      sin_port;   /* port dans l'ordre d'octets réseau */
    struct in_addr sin_addr;   /* adresse Internet                  */
};

/* Adresse Internet */
struct in_addr {
    u_int32_t      s_addr;     /* Adresse dans l'ordre d'octets réseau */
};
```

sin_family est toujours rempli avec **AF_INET**. C'est indispensable. Sous Linux 2.2, la plupart des fonctions réseau renvoient **EINVAL** lorsque cette configuration manque. *sin_port* contient le numéro de port, dans l'ordre des octets du réseau. Les numéros de ports inférieures à 1024 sont dits *réservés*. Seuls les processus privilégiés (c'est à dire ceux qui ont la capacité **CAP_NET_BIND_SERVICE**) peuvent appeler

bind(2) pour ces ports. Notez que le protocole IPv4 en tant que tel n'a pas de concept de ports, ils sont seulement implémentés par des protocoles de plus haut-niveau comme **tcp**(7) et **udp**(7).

sin_addr est l'adresse IP de l'hôte. le membre *s_addr* de la structure *struct in_addr* contient l'adresse de l'interface de l'hôte, dans l'ordre des octets du réseau. *in_addr* devrait être affecté par l'une des valeurs INADDR_* (par exemple, INADDR_ANY) ou configuré avec les fonctions de bibliothèque **inet_aton**(3), **inet_addr**(3), **inet_makeaddr**(3) ou directement par le système de résolution des noms (voir **gethostbyname**(3)). Les adresses IPv4 sont divisées en adresses unicast, broadcast et multicast. Les adresses unicast décrivent une interface unique d'un hôte, les adresses broadcast correspondent à tous les hôtes d'un réseau, et les adresses multicast représentent tous les hôtes d'un groupe multicast. Les datagrammes vers des adresses broadcast ne peuvent être émis et reçus que si l'attribut de socket **SO_BROADCAST** est activé. Dans l'implémentation actuelle, les sockets orientées connexion ne sont autorisées que sur des adresses unicast.

Remarquez que l'adresse et le port sont toujours stockés dans l'ordre des octets du réseau. Cela signifie qu'il faut invoquer **htons**(3) sur le numéro attribué à un port. Toutes les fonctions de manipulation d'adresse et port de la bibliothèque standard fonctionne dans l'ordre du réseau.

Il existe plusieurs adresses particulières : **INADDR_LOOPBACK** (127.0.0.1) correspond toujours à l'hôte local via le périphérique loopback ; **INADDR_ANY** (0.0.0.0) signifie un attachement à n'importe quelle adresse ; **INADDR_BROADCAST** (255.255.255.255) signifie n'importe quel hôte et à le même effet que **INADDR_ANY** pour des raisons historiques.

OPTIONS DES SOCKETS

IP supporte quelques options des sockets spécifiques aux protocoles, fixées avec **setsockopt**(2) et consultées avec **getsockopt**(2). Le niveau d'option de socket pour IP est **IPPROTO_IP**. Un attribut booléen en faux quand il est nul, et vrai sinon.

IP_OPTIONS

Fixe ou lit les options IP à envoyer avec chaque paquet sur cette socket. Les arguments sont un pointeur sur un tampon contenant les options et la longueur des options. L'appel **setsockopt**(2) fixe les options IP associées à une socket. La taille maximale des options pour IPv4 est 40 octets. Voir la RFC 791 pour les options autorisées. Lorsque le paquet de connexion initiale d'une socket **SOCK_STREAM** contient des options IP, celles-ci seront automatiquement attribué à la socket, avec les options de routage inversées. Les paquets entrants ne peuvent pas modifier les options après que la connexion est établie. Le traitement des options de routage des paquets entrant est désactivé par défaut, et peut être validé en utilisant la requête sysctl **accept_source_route**. Les autres options, comme l'horodatage sont toujours traitées. Pour les socket datagrammes, les options IP ne peuvent être fixées que par l'utilisateur local. L'appel de **getsockopt**(2) avec *IP_OPTIONS* remplit le tampon fourni avec les options d'émission actuelles.

IP_PKTINFO

Fournit un message *IP_PKTINFO* de service, qui contient une structure *pktinfo* fournissant quelques informations à propos du paquet entrant. Ceci ne fonctionne que pour les sockets orientées datagrammes. L'argument est un drapeau indiquant à la socket sur le message IP_PKTINFO doit être passé ou non. Le message lui-même ne peut être écrit ou lu que comme message de contrôle avec un paquet, en utilisant **recvmsg**(2) ou **sendmsg**(2).

```
struct in_pktinfo {
  unsigned int   ipi_ifindex;  /* Numéro d'interface  */
  struct in_addr ipi_spec_dst; /* Adresse locale      */
  struct in_addr ipi_addr;     /* Adresse destination */
};
```

ipi_ifindex est le numéro unique de l'interface sur laquelle le paquet a été reçu. *ipi_spec_dst* est l'adresse locale du paquet, et *ipi_addr* est l'adresse de destination dans l'en-tête du paquet. Si *IP_PKTINFO* est passé à **sendmsg**(2) et si *ipi_spec_dest* est non nul, alors il sera utilisé comme adresse source pour la recherche dans la table de routage, et pour fixer les options de routage IP.

Si *ipi_ifindex* est non nul, l'adresse localE principale de l'interface indiquée par cet index écrase *ipi_spec_dst* pour La table de routage.

IP_RECVTOS
Le message de service *IP_TOS* est passé avec les paquets entrants. Il contient un octet qui décrit le champ Type-Of-Service/Précédence de l'en-tête du paquet. Il s'agit d'un drapeau entier booléen.

IP_RECVTTL
Passer un message de contrôle *IP_TTL* avec le champ Time-To-Live du paquet reçu comme argument sous forme d'octet. Non supporté pour les sockets **SOCK_STREAM**.

IP_RECVOPTS
Passer à l'utilisateur toutes les options IP entrantes dans un message de contrôle *IP_OPTIONS*. L'en-tête de routage et les autres options sont déjà remplies pour l'hôte local. Non supporté pour les sockets *SOCK_STREAM*.

IP_RETOPTS
Comme *IP_RECVOPTS* mais renvoie les options non traitées, avec les options d'horodatage et de routage non remplies pour ce saut (hop).

IP_TOS
Fixe ou consulte le champs Type-Of-Service (TOS) envoyé avec chaque paquet IP sortant de cette socket. Il sert à gérer sur le réseau les priorités entre paquets. TOS est un octet. Quelques attributs TOS standards sont définis : **IPTOS_LOWDELAY** pour minimiser les délais en trafic interactif, **IPTOS_THROUGHPUT** pour optimiser le débit, **IPTOS_RELIABILITY** pour optimiser la fiabilité, **IPTOS_MINCOST** doit être utilisé pour les données de remplissage, quand la lenteur de transmission importe peu. Une de ces valeurs TOS au maximum peut être indiquée. Les autres bits sont invalides et doivent être effacés. Linux envoie d'abord des datagrammes **IPTOS_LOWDELAY** par défaut, mais le comportement exact dépend de la politique configurée pour la file d'attente. Quelques niveaux de haute priorité peuvent réclamer des privilèges super utilisateur (la capacité **CAP_NET_ADMIN**). La priorité peut aussi être indiquée d'une manière indépendante du protocole avec les options (**SOL_SOCKET**, **SO_PRIORITY**) de socket (voir **socket**(7)).

IP_TTL
Fixer ou consulter le contenu actuel du champ Time-To-Live envoyé avec chaque paquet sortant de cette socket.

IP_HDRINCL
L'utilisateur doit fournir un en-tête ip avant les données proprement dites. Uniquement valides pour les sockets **SOCK_RAW**. Voir **raw**(7) pour plus de détail. Lorsque cet attribut est activé, les valeurs fixées pour *IP_OPTIONS*, *IP_TTL* et *IP_TOS* sont ignorées.

IP_RECVERR (défini dans <linux/errqueue.h>)
Active le passage amélioré des messages d'erreur. Lorsque cet attribut est activé sur une socket datagramme, les erreurs seront mémorisées dans une file particulière pour la socket. Quand l'utilisateur détecte un échec d'une opération sur la socket, les erreurs peuvent être examinées en invoquant **recvmsg**(2) avec l'attribut **MSG_ERRQUEUE**. La structure *sock_extended_err* décrivant l'erreur sera passé comme message de service ayant le type *IP_RECVERR* et le niveau **IPPROTO_IP**. Ceci permet une gestion d'erreur fiable sur les sockets non connectées. La partie "données reçues" de la file d'erreurs contient le paquet ayant rencontré un problème.

Le message de contrôle *IP_RECVERR* contient une structure *sock_extended_err* :

```
#define SO_EE_ORIGIN_NONE    0
#define SO_EE_ORIGIN_LOCAL   1
#define SO_EE_ORIGIN_ICMP    2
#define SO_EE_ORIGIN_ICMP6   3

struct sock_extended_err {
  u_int32_t ee_errno;   /* numéro d'erreur */
  u_int8_t  ee_origin;  /* origine de l'erreur */
  u_int8_t  ee_type;    /* type */
  u_int8_t  ee_code;    /* code */
  u_int8_t  ee_pad;
  u_int32_t ee_info;    /* autres informations */
  u_int32_t ee_data;    /* autres données */
  /* champs supplémentaires éventuels */
};

struct sockaddr *SOCK_EE_OFFENDER(struct sock_extended_err *);
```

ee_errno contient le numéro de l'erreur mise en file. *ee_origin* est le code de l'origine de l'erreur. Les autres champs sont spécifiques au protocole. La macro **SOCK_EE_OFFENDER** renvoie un pointeur sur l'adresse d'un objet réseau d'où l'erreur provient, en prenant en argument un pointeur sur le message de service. Si cette adresse n'est pas disponible, le membre *sa_family* de la structure *sockaddr* contient **AF_UNSPEC** et les autres champs de *sockaddr* sont indéfinis.

IP utilise la structure *sock_extended_err* comme suit : *ee_origin* contient **SO_EE_ORIGIN_ICMP** pour les erreurs reçues sous forme de paquet ICMP, ou **SO_EE_ORIGIN_LOCAL** pour les erreurs locales. Les valeurs inconnues doivent être ignorées. *ee_type* et *ee_code* sont remplis avec les champs type et code de l'en-tête ICMP. *ee_info* contient le MTU déterminé pour les erreurs **EMSGSIZE**. Le message contient aussi l'adresse *sockaddr_in* du noeud ayant causé l'erreur, qui peut être obtenu avec la macro. **SOCK_EE_OFFENDER**. Le champ *sin_family* de l'adresse fournie par SOCK_EE_OFFENDER vaut *AF_UNSPEC* si la source était inconnue. Lorsque les erreurs proviennent du réseau, toutes les options IP (*IP_OPTIONS*, *IP_TTL*, etc.) valides pour la socket, et contenues dans le paquet en erreur sont transmises comme messages de contrôle. Le contenu original du paquet causant l'erreur est renvoyé comme charge normale. Notez que TCP n'a pas de file d'erreurs ; **MSG_ERRQUEUE** est illégal sur les sockets **SOCK_STREAM**. Ainsi, toutes les erreurs sont renvoyées par les fonctions sur les sockets ou par **SO_ERROR** seulement.

Pour les sockets raw, *IP_RECVERR* valide le passage de toutes les erreurs ICMP reçues à l'application, sinon les erreurs sont seulement renvoyées sur les sockets connectées. Il s'agit d'un attribut booléen entier. *IP_RECVERR* est désactivé par défaut.

IP_MTU_DISCOVER

Fixe ou consulte l'attribut de recherche du MTU du chemin (Path MTU - PMTU) pour une socket. Lorsqu'il est activé, Linux effectuer la recherche du MTU de chemin comme défini dans la RFC 1191. L'attribut interdisant la fragmentation est alors activé sur tous les datagrammes sortants. La valeur par défaut est commandée au niveau système par le sysctl **ip_no_pmtu_disc** pour les sockets **SOCK_STREAM**, et désactivé pour toutes les autres. Pour les sockets autres que **SOCK_STREAM** il est de la responsabilité de l'utilisateur d'empaqueter les données dans des blocs inférieurs au MTU et d'assurer la retransmission si besoin est. Le noyau rejettera les paquets qui sont plus gros que le MTU déterminé si cet attribut est activé (avec l'erreur **EMSGSIZE**).

Attribut MTU chemin	Signification
IP_PMTUDISC_WANT	Utiliser une configuration par route.
IP_PMTUDISC_DONT	Ne pas rechercher le MTU par chemin.

IP_PMTUDISC_DO Toujours rechercher le MTU par chemin.

Lorsque la recherche du PMTU est active, le noyau garde automatiquement une trace des MTU des chemins par hôte destinataire. Lorsqu'il est connecté à un correspondant spécifique avec **connect**(2), le MTU du chemin actuellement déterminé peut être consulté en utilisant l'option **IP_MTU** de la socket (par exemple si une erreur **EMSGSIZE** se produit). Cette valeur peut changer dans le temps. Pour les sockets sans connexions, avec plusieurs destinations, le nouveau MTU pour une destination donnée peut également être obtenu en utilisant la file d'erreur (voir **IP_RECVERR**). Une nouvelle erreur sera mise en file pour chaque mise à jour du MTU.

Durant la recherche du MTU, les paquets initiaux des sockets datagrammes peuvent être perdus. Les applications utilisant UDP devraient le savoir, et les éviter dans leur stratégie de retransmission.

Pour démarrer le processus de recherche du MTU par chemin sur les sockets non-connectées, il est possible de démarrer avec une grande taille de datagramme (jusqu'à 64 ko d'en-tête) et la diminuer au fur et à mesure des mises à jours du MTU du chemin.

Pour obtenir une estimation initiale du MTU d'un chemin connectez une socket datagramme à l'adresse de destination en utilisant **connect**(2) et consultez le MTU en appelant **getsockopt**(2) avec l'option **IP_MTU**.

IP_MTU
Renvoie le MTU du chemin actuellement déterminé pour la socket. Seulement valide quand la socket a été connectée. Renvoie un entier. Uniquement valide pour un **getsockopt**(2).

IP_ROUTER_ALERT
Passer tous les futurs paquets redirigés (forwarded) avec l'option IP Router Alert activée sur cette socket. Uniquement valide pour les sockets raw. Ceci sert par exemple pour les démons RSVP de l'espace utilisateur. Les paquets enregistrés ne sont pas redirigés par le noyau, c'est la responsabilité de l'utilisateur de les renvoyer. L'attachement des sockets est ignoré, et de tels paquets ne sont filtrés que par le protocole. Il s'agit d'un attribut entier.

IP_MULTICAST_TTL
Fixe ou consulte la valeur du champs Time-To-Live des paquets multicast sortant sur cette socket. Il est très importants pour les paquets multicast de fixer le TTL le plus petit possible. La valeur par défaut est 1, ce qui signifie que les paquet multicast ne quittent pas le réseau local à moins que le programme de l'utilisateur ne le réclame explicitement. L'argument est un entier.

IP_MULTICAST_LOOP
Lit ou écrit un entier booléen indiquant si les paquets multicast doivent être renvoyés en boucle aux sockets locales concernées.

IP_ADD_MEMBERSHIP
Rejoindre un groupe multicast. L'argument est une structure *struct ip_mreqn*.

```
struct ip_mreqn {
   struct in_addr imr_multiaddr; /* Adresse IP du groupe multicast */
   struct in_addr imr_address;   /* Adresse IP de l'interface locale */
   int           imr_ifindex;   /* Numéro d'interface */
};
```

imr_multiaddr contient l'adresse du groupe multicast que l'application veut rejoindre ou quitter. Il doit s'agir d'une adresse multicast valide. *imr_address* est l'adresse de l'interface locale avec laquelle le système doit joindre le groupe multicast. Si elle est égale à **INADDR_ANY**, une interface appropriée est choisie par le système. *imr_ifindex* est le numéro de l'interface pour rejoindre ou quitter le groupe *imr_multiaddr*, ou zéro pour indiquer n'importe quelle interface.

Pour la compatibilité, l'ancienne structure *ip_mreq* est encore supportée. Elle diffère de *ip_mreqn* seulement par l'absence du membre *imr_ifindex*. Uniquement valide pour **setsockopt**(2).

IP_DROP_MEMBERSHIP
Quitter un groupe multicast. L'argument est une structure **ip_mreqn** ou **ip_mreq** comme pour *IP_ADD_MEMBERSHIP*.

IP_MULTICAST_IF
Fixer le périphérique local pour une socket multicast. L'argument est une structure *ip_mreqn* ou *ip_mreq* comme pour *IP_ADD_MEMBERSHIP*.

Lorsqu'une option de socket invalide est fournie, **ENOPROTOOPT** est renvoyée.

SYSCTLS

Le protocole IP support l'interface sysctl pour configurer certaines options globales. Les sysctl peuvent être réalisés en lisant ou écrivant dans les fichiers */proc/sys/net/ipv4/** ou en utilisant l'interface **sysctl**(2). Les variables indiquées *booléen* prennent une valeur entière, avec une valeur non nulle (vrai) signifiant que l'option correspondante est activée et une valeur nulle (faux) signifiant que l'option est déactivée.

ip_always_defrag (booléen)
[Nouveauté des noyaux 2.2.13, dans les noyaux précédents c'était une option de compilation nommée **CONFIG_IP_ALWAYS_DEFRAG** ; ce fichier n'est plus présent dans les noyaux 2.4.x et ultérieurs]

Lorsque ce drapeau booléen et actif (différent de zéro), les fragments entrants (morceaux de paquets IP obtenus car un hôte entre l'origine et la destination a décidé que les paquets étaient trop grands et les a coupé en morceaux) seront réassemblés (défragmentés) avant d'être traités, même s'ils doivent être redirigés (forwarded).

À utiliser uniquement pour un firewall qui est le seul lien d'entrée de votre réseau, ou un proxy transparent. Ne jamais activer pour un routeur normal ou un hôte. Sinon, les communications fragmentées peuvent être interrompues lorsque les fragments circulent par différents liens. La défragmentation a également un coût mémoire et CPU non négligeable.

Ceci est automatiquement activé lorsque le masquerading ou le proxy transparent est configuré.

ip_autoconfig
Non documenté.

ip_default_ttl (entier ; valeur par défaut : 64)
Fixe la valeur par défaut du champ Time-To-Live des paquets sortants. Ceci peut être modifié individuellement pour chaque socket avec l'option *IP_TTL*.

ip_dynaddr
Active la réécriture dynamique des adresses de socket et du masquerading lors des changements d'adresse d'interface. Cela sert pour les liaisons par modem, avec des adresses IP variables. 0 signifie aucune réécriture, 1 les autorise, et 2 demande un mode volubile.

ip_forward (booléen ; valeur par défaut : désactivé)
Active la redirection IP (forwarding) avec un attribut booléen. La redirection IP peut aussi être configurée interface par interface.

ip_local_port_range
Contient deux entiers qui définissent l'intervalle par défaut des ports locaux alloués aux sockets. L'allocation démarre avec le premier numéro et se termine avec le second. Notez que cela ne doit pas entrer en conflit avec les ports utilisés pour le masquerading (bien que cela soit traité). De même des choix arbitraires peuvent poser des problèmes avec certains firewalls de filtrage par paquet qui font des suppositions sur les ports locaux utilisés. Le premier nombre doit être au moins supérieur à 1024 et de préférence à 4096 pour éviter les collisions avec les ports officiels et minimiser les problèmes de firewall.

ip_no_pmtu_disc (booléen ; valeur par défaut : désactivé)
Désactiver la recherche par défaut des MTU par chemin pour les sockets TCP. La recherche du MTU par chemin peut échouer avec des firewalls mal configurés (qui rejettent tous les paquets ICMP) ou les interfaces mal configurées (par exemple lien point-à-point où les deux extrémités n'ont pas le même MTU). Il vaut mieux corriger le routeur défectueux que de supprimer globalement la recherche du MTU par chemin, car cette dernière option augmente les coûts du réseau. +. +. +.TP +.BR ip_nonlocal_bind " (booléen ; valeur par défaut : désactivé)" Si elle est activée, elle permet aux processus d'associer (**bind**()) des adresses IP qui ne soient pas locales, ce qui peut être utiles mais faire planter certaines applications.

ip6frag_time (entier ; valeur par défaut : 30)
La durée, en secondes, de conservasion d'un fragment IPv6 en mémoire.

ip6frag_secret_interval (Entier ; valeur par défaut : 30)
Intervalle de regénération (en secondes) de l'empreinte secrète (ou temps de vie de l'empreinte secrète) des fragments IPv6.

ipfrag_high_thresh (entier), **ipfrag_low_thresh** (entier)
Si le nombre de fragments IP en file atteint **ipfrag_high_thresh**, la file est restreinte à **ipfrag_low_thresh**. Contient un entier avec le nombre d'octets.

neigh/*
voir **arp**(7).

IOCTLS

Toutes les ioctls décrites dans **socket**(7) s'appliquent à la couche IP.

Les ioctls pour configurer les paramètres génériques des périphériques sont décrits dans **netdevice**(7).

NOTES

Soyez très prudents avec l'option **SO_BROADCAST**, elle n'est pas privilégiée sous Linux. Il est facile de surcharger un réseau avec des broadcast sans précaution. Pour les nouveaux protocoles applicatifs, il vaut mieux utiliser un groupe multicast plutôt que le broadcast. Ce dernier est déconseillé.

Certaines autres implémentations des sockets BSD fournissent les options de socket *IP_RCVDSTADDR* et *IP_RECVIF* pour obtenir l'adresse de destination et l'interface des datagrammes reçus. Linux à l'option *IP_PKTINFO* plus générale pour effectuer ce travail.

Certaines implémentations de sockets BSD fournissent également une option *IP_RECVTTL*, mais un message auxilliaire de type *IP_RECVTTL* est passé avec le paquet entrant. Cela est différent de l'option *IP_TTL* sous Linux.

L'utilisation du niveau d'option socket *SOL_IP* n'est pas portable, les piles basées sur BSD utilisent le niveau *IPPROTO_IP*.

ERREURS

ENOTCONN
L'opération n'est définie que pour une socket connectée, mais la socket ne l'était pas.

EINVAL
Un argument invalide a été transmis. Pour les émissions, cela peut être causé par un envoi vers une route *trou noir*.

EMSGSIZE
Datagramme plus grand que le MTU du chemin, et ne peut pas être fragmenté.

EACCES
L'utilisateur essaye de réaliser une opération sans avoir les permissions nécessaires. Cela inclut : L'envoi d'un paquet vers une adresse broadcast sans avoir activé l'attribut **SO_BROADCAST**. L'envoi d'un paquet vers une route *interdite*. Modification du paramétrage du firewall sans privilèges super utilisateur (la capacité **CAP_NET_ADMIN**). Attachement à un port réservé sans privilèges super utilisateur (la capacité **CAP_NET_BIND_SERVICE**).

EADDRINUSE
Tentative d'attachement à une adresse déjà utilisée.

ENOPROTOOPT et **EOPNOTSUPP**
Passage d'une option de socket invalide.

EPERM
L'utilisateur n'a pas la permission de fixer une priorité haute, de changer la configuration ou d'envoyer des signaux au groupe ou au processus demandé.

EADDRNOTAVAIL
Une interface inexistante ou une adresse source non locale ont été réclamées.

EAGAIN
L'opération sur une socket non-bloquante devrait bloquer.

ESOCKTNOSUPPORT
La socket n'est pas configurée ou on a demandé un type de socket inconnu.

EISCONN
connect(2) a été appelé sur une socket déjà connectée.

EALREADY
Une opération de connexion est déjà en cours sur une socket non-bloquante.

ECONNABORTED
Une connexion a été fermée durant un **accept**(2).

EPIPE La connexion a été fermée prématurément ou terminée par le correspondant.

ENOENT
SIOCGSTAMP a été appelé sur une socket sans qu'aucun paquet n'y soit disponible.

EHOSTUNREACH
Aucune route valide dans la table ne correspond à l'adresse de destination. Cette erreur peut être due à un message ICMP d'un routeur distant ou à la table de routage interne.

ENODEV
Le périphérique réseau n'est pas disponible ou est incapable d'envoyer de l'IP.

ENOPKG
Un sous-système du noyau n'est pas configuré.

ENOBUFS, **ENOMEM**
Pas assez de mémoire. Cela signifie souvent que l'allocation mémoire est contrainte par les limites du tampon de socket, pas par la mémoire système, mais ce n'est pas toujours sûr.

D'autres erreurs peuvent être déclenchées par les protocoles supérieurs. Voir **tcp**(7), **raw**(7), **udp**(7) et **socket**(7).

VERSIONS

IP_MTU, *IP_MTU_DISCOVER*, *IP_PKTINFO*, *IP_RECVERR* et *IP_ROUTER_ALERT* sont de nouvelles options de Linux 2.2. Elles sont aussi spécifiques à Linux et ne doivent pas servir dans des programmes destinés à être portables.

struct ip_mreqn est nouvelle dans Linux 2.2. Sous Linux 2.0, seule existait *ip_mreq*.

Les sysctls ont été introduits avec Linux 2.2.

COMPATIBILITÉ

Pour la compatibilité avec Linux 2.0, la syntaxe obsolète **socket(PF_INET, SOCK_PACKET,** *protocole***)** est encore supportée pour ouvrir une socket **packet**(7). Cela est déconseillé, et doit être remplacé par un **socket(PF_PACKET, SOCK_RAW,** *protocole***)** La principale différence est la nouvelle structure d'adresse *sockaddr_ll* pour les informations génériques du niveau liaison à la place de l'ancienne *sockaddr_pkt*.

BOGUES

Il y a trop de valeurs d'erreurs hétérogènes.

Les ioctls pour configurer les options d'interface spécifiques IP et les tables ARP ne sont pas décrites.

Certaines versions de la GlibC oublient la déclaration *in_pktinfo*. Le remède est de recopier dans votre programme la description de cette page de manuel.

La réception de l'adresse de destination originale avec **MSG_ERRQUEUE** dans *msg_name* par **recvmsg**(2) ne fonctionne pas dans certains noyaux 2.2.

VOIR AUSSI

recvmsg(2), **sendmsg**(2), **byteorder**(3), **ipfw**(4), **capabilities**(7), **netlink**(7), **raw**(7), **socket**(7), **tcp**(7), **udp**(7)

RFC 791 pour les spécifications IP d'origine.
RFC 1122 pour les nécessités IPv4 des hôtes.
RFC 1812 pour les nécessités IPv4 des routeurs.

TRADUCTION

Ce document est une traduction réalisée par Christophe Blaess <http://www.blaess.fr/christophe/> le 9 juin 2001 et révisée le 11 août 2006.

L'équipe de traduction a fait le maximum pour réaliser une adaptation française de qualité. La version anglaise la plus à jour de ce document est toujours consultable via la commande : « **LANG=en man 7 ip** ». N'hésitez pas à signaler à l'auteur ou au traducteur, selon le cas, toute erreur dans cette page de manuel.